

Non-termination using Regular Languages*

— International Workshop on Termination —

Jörg Endrullis¹ and Hans Zantema²

¹ VU University Amsterdam, The Netherlands

² Eindhoven University of Technology, The Netherlands

Abstract

We describe a method for proving non-termination of term rewriting systems that do not admit looping reductions. As certificates of non-termination, we employ regular (tree) automata.

1998 ACM Subject Classification D.1.1, D.3.1, F.4.1, F.4.2, I.1.1, I.1.3

Keywords and phrases non-termination, finite automata, regular languages

1 Introduction

We describe a method for proving non-termination of term rewriting systems that do not admit looping reductions, that is, reductions from a term t to a term $C[t\sigma]$ containing a substitution instance of t . For this purpose, we employ tree automata as certificates of non-termination. For proving non-termination of a term rewriting system R , we search a tree automaton A whose language $\mathcal{L}(A)$ is not empty, weakly closed under rewriting and every term of the language contains a redex occurrence. We have fully automated the search for these certificates employing SAT-solvers.

All automata that we use as example in this paper have been found automatically; this concerns in particular fully automated proofs of non-termination for the following two rewrite systems.

► **Example 1.** We consider the following string rewriting system:

$$zL \rightarrow Lz \qquad Rz \rightarrow zR \qquad bL \rightarrow bR \qquad Rb \rightarrow Lzb$$

This rewrite system admits no reductions of the form $s \rightarrow^* \ell sr$.

► **Example 2.** We consider the S -rule from combinatory logic:

$$ap(ap(ap(S, x), y), z) \rightarrow ap(ap(x, z), ap(y, z))$$

For the S -rule it is known that there are no reductions $t \rightarrow^* C[t]$ for ground terms t , see [15]. For open terms t the existence of reductions $t \rightarrow^* C[t\sigma]$ is open.

It turns out that the method can be fruitfully applied to obtain non-termination proofs of several string rewriting systems that have remained unsolved in the last full run of the termination competition.

* Published at the International Workshop on Termination 2014.



Related Work

The paper [11] investigates necessary conditions for the existence of loops. The work [17] employs SAT solvers to find loops, [18] uses forward closures to find loops efficiently, and the work [16] introduces ‘compressed loops’ to find certain forms of (possibly very long) loops.

Non-termination beyond loops has been investigated in [14] and [2]; we note that Example 2 cannot be handled by these techniques.

Here we prove non-looping non-termination on regular languages. The converse, local termination on regular languages, has been investigated in [3]. Regular (tree) automata have been fruitfully applied to a wide range of properties of term rewriting systems: for proving termination [10, 8, 12], for infinitary normalization [4], for proving liveness [13], and for analysing reachability and deciding the existence of common reducts [9, 5].

2 Non-termination and Weakly Closed Languages

- **Definition 3.** Let $L \subseteq T(\Sigma, \emptyset)$ a language and R a TRS over Σ . Then L is called:
- *closed* under rewriting if for every $t \in L$ and s such that $t \rightarrow s$, one has $s \in L$, and
 - *weakly closed* under rewriting if for every $t \in L$ that is not in normal form, there exists $s \in L$ such that $t \rightarrow_R s$.

The following theorem describes the basic idea that we employ for proving non-termination.

- **Theorem 4.** *A term rewriting system R over Σ is non-terminating if and only if there exists a non-empty language $L \subseteq T(\Sigma, \mathcal{X})$ such that*

- (i) *every $t \in L$ contains a redex (that is, $t \rightarrow s$ for some term s), and*
- (ii) *L is weakly closed under rewriting.* ◀

A language fulfilling the properties of Theorem 4 is also called a *recurrence set*, see [1].

To automate this method, we need to restrict to a certain family of languages. In this paper, we consider regular tree languages. To guarantee that the language of a tree automaton is weakly closed under rewriting, we check that the language is not empty and that the automaton is a quasi-model (see Definition 13) for the rewrite system. The latter condition is actually too strict; it implies that the languages is not only weakly closed, but also closed under rewriting. In future, we plan to relieve this restriction.

3 Tree Automata

- **Definition 5.** A (nondeterministic finite) tree automaton A over a signature Σ is a tuple $A = \langle Q, \Sigma, F, \delta \rangle$ where

- (i) Q is a finite set of *states*,
- (ii) $F \subseteq Q$ is a set of *accepting states*, and
- (iii) $\{\delta_f\}_{f \in \Sigma}$ is a family of *transition relations* such that for every $f \in \Sigma$:

$$\delta_f \subseteq Q^n \times Q$$

where n is the arity of f .

In examples, we often write the transition relation δ_f as \rightarrow_f .

► **Example 6.** The following is a tree automaton for the signature in Example 1. We consider string rewriting systems as term rewriting systems by interpreting all symbols as unary and adding a special constant ε to denote the end of the word. Let $A_{LR} = \langle Q, \Sigma, F, \rightarrow \rangle$ where $Q = \{0, 1, 2, 3\}$, $\Sigma = \{b, L, R, 0, \varepsilon\}$, $F = \{3\}$ and

$$\begin{array}{ccccc} \rightarrow_\varepsilon 0 & 1 \rightarrow_z 1 & 0 \rightarrow_b 1 & 1 \rightarrow_R 2 & 1 \rightarrow_L 2 \\ & 2 \rightarrow_z 2 & 2 \rightarrow_b 3 & & \end{array}$$

The transition relation for ε can be thought of as defining the initial states (here 0) of a word automaton.

► **Example 7.** The following is a tree automaton for Example 2. Let $A_S = \langle Q, \Sigma, F, \rightarrow \rangle$ where $Q = \{0, 1, 2, 3, 4\}$, $\Sigma = \{ap, S\}$, $F = \{4\}$ and

$$\begin{array}{ccccc} \rightarrow_S 0 & (0, 0) \rightarrow_{ap} 1 & (1, 0) \rightarrow_{ap} 2 & (2, 2) \rightarrow_{ap} 3 & (3, 3) \rightarrow_{ap} 3 \\ & (0, 2) \rightarrow_{ap} 2 & & (2, 3) \rightarrow_{ap} 3 & (3, 3) \rightarrow_{ap} 4 \\ & (0, 3) \rightarrow_{ap} 2 & & & \end{array}$$

In Example 12 we show that this automaton accepts the term $SSS(SSS)(SSS(SSS))$.

► **Definition 8.** Let $A = \langle Q, \Sigma, F, \delta \rangle$ be a tree automaton over Σ . For terms $t \in T(\Sigma, \mathcal{X})$ and assignments $\alpha : \mathcal{X} \rightarrow \mathcal{P}(Q)$ we define the *interpretation* $[t, \alpha]_A$ by:

$$\begin{aligned} [x, \alpha]_A &= \alpha(x) \\ [f(t_1, \dots, t_n), \alpha]_A &= \{q \mid (q_1, \dots, q_n) \in [t_1, \alpha]_A \times \dots \times [t_n, \alpha]_A, \langle (q_1, \dots, q_n), q \rangle \in \delta_f\} \end{aligned}$$

Whenever A is clear from the context, we write $[t, \alpha]$ as shorthand for $[t, \alpha]_A$. For ground terms t , the interpretation is independent of α , allowing us to write $[t]_A$ or $[t]$ for short.

► **Example 9.** We use the automaton A_S from Example 7. Let $\alpha(x) = \{2\}$, then we have:

$$\begin{aligned} [S, \alpha] &= \{0\} & [ap(S, S), \alpha] &= \{1\} & [ap(ap(S, S), S), \alpha] &= \{2\} \\ [ap(x, x), \alpha] &= \{3\} & [ap(ap(x, x), ap(x, x)), \alpha] &= \{3, 4\} \end{aligned}$$

► **Definition 10.** Let $A = \langle Q, \Sigma, F, \delta \rangle$ be a tree automaton over Σ . The *language* $\mathcal{L}(A)$ accepted by A is the set $\mathcal{L}(A) = \{t \mid t \in T(\Sigma, \emptyset), [t]_A \cap F \neq \emptyset\}$ of ground terms.

► **Example 11.** The automaton in Example 6 accepts all words of the form $b z^* (L|R) z^* b$, that is, all words that start with b , end with b , contain one L or R and otherwise only z .

► **Example 12.** We continue Example 9:

$$\begin{aligned} [ap(ap(S, S), S)] &= \{2\} & [ap(ap(ap(S, S), S), ap(ap(S, S), S))] &= \{3\} \\ [ap(ap(ap(ap(S, S), S), ap(ap(S, S), S)), ap(ap(ap(S, S), S), ap(ap(S, S), S)))] &= \{3, 4\} \end{aligned}$$

Thus $F \cap [SSS(SSS)(SSS(SSS))] = \{4\} \neq \emptyset$ and hence the term is accepted by the automaton.

4 Closure under Rewriting

► **Definition 13.** A tree automaton $A = \langle Q, \Sigma, F, \delta \rangle$ is a *quasi-model* for a term rewriting system R over Σ if $[\ell, \alpha]_A \subseteq [r, \alpha]_A$ for every $\ell \rightarrow r \in R$ and $\alpha : \mathcal{X} \rightarrow \mathcal{P}(Q)$.

Actually, it suffices to check the property $[\ell, \alpha]_A \subseteq [r, \alpha]_A$ for assignments $\alpha : \mathcal{X} \rightarrow \mathcal{P}(Q)$ that map variables to singleton sets.

► **Lemma 14.** *A tree automaton $A = \langle Q, \Sigma, F, \delta \rangle$ is a quasi-model for a term rewriting system R over Σ iff $[\ell, \alpha]_A \subseteq [r, \alpha]_A$ for every $\ell \rightarrow r \in R$ and $\alpha : \mathcal{X} \rightarrow \{\{q\} \mid q \in Q\}$.*

► **Example 15.** It is not difficult to check that the automaton A_{LR} from Example 6 is a quasi-model for rewrite system in Example 1.

► **Example 16.** We consider the automaton A_S from Example 7. We write $(a, b, c) \rightarrow d$ if $d \in [\ell, \alpha]$ when $\alpha(x) = \{a\}$, $\alpha(y) = \{b\}$, $\alpha(z) = \{c\}$. Then for $[\ell, \alpha]$ we have:

$$\begin{array}{lllll} (0, 0, 2) \rightarrow 1 & (2, 2, 3) \rightarrow 3 & (2, 3, 3) \rightarrow 3 & (3, 2, 3) \rightarrow 3 & (3, 3, 3) \rightarrow 3 \\ (0, 0, 3) \rightarrow 1 & (2, 2, 3) \rightarrow 4 & (2, 3, 3) \rightarrow 4 & (3, 2, 3) \rightarrow 4 & (3, 3, 3) \rightarrow 4 \end{array}$$

The interpretation $[r, \alpha]$ has all the above and additionally:

$$\begin{array}{lll} (0, 2, 2) \rightarrow 3 & (1, 1, 0) \rightarrow 3 & (2, 2, 2) \rightarrow 3 \\ (0, 2, 3) \rightarrow 3 & & (2, 2, 2) \rightarrow 4 \\ (0, 3, 3) \rightarrow 3 & & \end{array}$$

As a consequence A_S is a quasi-model for the S -rule.

The following theorem is immediate:

► **Theorem 17.** *Let $A = \langle Q, \Sigma, F, \delta \rangle$ be a tree automaton and R a term rewriting system over Σ . If A is a quasi-model for R then the language of A is closed under rewriting.* ◀

5 Ensuring Redex Occurrences

Next, we want to guarantee that every term in the language $\mathcal{L}(A)$ of an automaton A contains a redex with respect to the term rewriting system R . For left-linear systems R , this problem can be reduced to deciding the inclusion of regular languages.

Let R be a left-linear term rewriting system. Then the set of ground terms containing a redex is a regular tree language. A deterministic automaton B for this language can be constructed using the overlap-closure of subterms of left-hand sides, see further [6, 7].

► **Example 18.** The following tree automaton $C = \langle Q, \Sigma, F, \rightarrow \rangle$ accepts the language of ground terms that contain a redex occurrence with respect to the S -rule. Here $Q = \{0, 1, 2, 3\}$, $\Sigma = \{ap, S\}$, $F = \{3\}$ and

$$\rightarrow_S 0 \quad (0, q) \rightarrow_{ap} 1 \quad (1, q) \rightarrow_{ap} 2 \quad (2, q) \rightarrow_{ap} 3 \quad (3, q) \rightarrow_{ap} 3 \quad (q, 3) \rightarrow_{ap} 3$$

for all $q \in \{0, 1, 2\}$.

As a consequence the problem of checking whether every term in $\mathcal{L}(A)$ contains a redex boils down to checking that $\mathcal{L}(A) \subseteq \mathcal{L}(B)$. For non-deterministic A and deterministic B , this property can be decided by constructing the product automaton and considering the reachable states.

► **Definition 19.** The *product* $A \cdot B$ of tree automata $A = \langle Q, \Sigma, F, \delta \rangle$ and $B = \langle Q', \Sigma, F', \delta' \rangle$ is the automaton $C = \langle Q \times Q', \Sigma, \emptyset, \gamma \rangle$ where for every $f \in \Sigma$ of arity n , we define the transition relation $\gamma_f \subseteq (Q \times Q')^n \times (Q \times Q')$ by

$$\langle (q_1, p_1), \dots, (q_n, p_n) \rangle \gamma_f (q', p') \iff \langle q_1, \dots, q_n \rangle \delta_f q' \wedge \langle p_1, \dots, p_n \rangle \delta'_f p'$$

► **Definition 20.** The set of *reachable states* of a tree automaton $A = \langle Q, \Sigma, F, \delta \rangle$ is the smallest set $S \subseteq Q$ such that $q \in S$ whenever $\langle q_1, \dots, q_n \rangle \delta_f q$ for some $q_1, \dots, q_n \in S$ and $f \in \Sigma$ with arity n .

The following theorem gives a method for checking $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ without the need for determinising A (only B needs to be deterministic).

► **Theorem 21.** Let $A = \langle Q, \Sigma, F, \delta \rangle$ and $B = \langle Q', \Sigma, F', \delta' \rangle$ be tree automata such that B is deterministic. Let S be the set of reachable states of the product automaton $A \cdot B$. Then $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ if and only if for all $(q, p) \in S$ it holds that $q \in F \implies p \in F'$.

► **Example 22.** The reachable states of product automaton $A_S \cdot C$ of the automata A_S from Example 7 and C from Example 18 are $(0, 0), (1, 1), (2, 2), (2, 1), (3, 3), (3, 2), (2, 3), (4, 3)$. The only state (q, q') such that q is accepting in A_S is $(4, 3)$ and 3 is an accepting state of C . Thus the conditions of Theorem 21 are fulfilled and hence $\mathcal{L}(A_S) \subseteq \mathcal{L}(C)$. Thus every term accepted by A_S contains a redex.

6 Future Work

We plan to investigate whether the method described in this paper can be fruitfully extended from regular automata to pushdown automata, that is, context-free languages. For this purpose, it is important that it is decidable whether a context-free language is a subset of a regular language (the language of terms containing left-linear redex occurrences). However, it remains to be investigated whether context-free certificates can be found efficiently.

References

- 1 B. Cook. Principles of program termination. <http://research.microsoft.com/en-us/um/cambridge/projects/terminator/principles.pdf>.
- 2 F. Emmes, T. Enger, and J. Giesl. Proving Non-looping Non-termination Automatically. In *International Joint Conference on Automated Reasoning (IJCAR 2012)*, volume 7364 of *Lecture Notes in Computer Science*, pages 225–240. Springer, 2012.
- 3 J. Endrullis, R.C. de Vrijer, and J. Waldmann. Local Termination: Theory and Practice. *Logical Methods in Computer Science*, 6(3), 2010.
- 4 J. Endrullis, C. Grabmayer, D. Hendriks, J.W. Klop, and R.C. de Vrijer. Proving Infinitary Normalization. In *Postproc. Int. Workshop on Types for Proofs and Programs (TYPES 2008)*, volume 5497 of *Lecture Notes in Computer Science*, pages 64–82. Springer, 2009.
- 5 J. Endrullis, C. Grabmayer, J.W. Klop, and V. van Oostrom. On Equal μ -Terms. *Theoretical Computer Science*, 412(28):3175–3202, 2011.
- 6 J. Endrullis and D. Hendriks. Transforming Outermost into Context-Sensitive Rewriting. *Logical Methods in Computer Science*, 6(2), 2010.
- 7 J. Endrullis and D. Hendriks. Lazy Productivity via Termination. *Theoretical Computer Science*, 412(28):3203–3225, 2011.
- 8 J. Endrullis, D. Hofbauer, and J. Waldmann. Decomposing Terminating Rewrite Relations. In *Proc. Workshop on Termination (WST '06)*, pages 39–43, 2006.
- 9 B. Felgenhauer and R. Thiemann. Reachability Analysis with State-Compatible Automata. In *Proc. Conf. on Language and Automata Theory and Applications (LATA 2014)*, volume 8370 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 2014.
- 10 A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Inf. Comput.*, 205(4):512–534, 2007.
- 11 A. Geser and H. Zantema. Non-looping string rewriting. *ITA*, 33(3):279–302, 1999.

- 12 M. Korp and A. Middeldorp. Match-bounds revisited. *Inf. Comput.*, 207(11):1259–1283, 2009.
- 13 M. Mousazadeh, B. T. Ladani, and H. Zanema. Liveness verification in trss using tree automata and termination analysis. *Computing and Informatics*, 29(3):407–426, 2010.
- 14 M. Oppelt. Automatische Erkennung von Ableitungsmustern in nichtterminierenden Wortersetzungs-systemen. Technical report, HTWK Leipzig, Germany, 2008. Diploma Thesis.
- 15 J. Waldmann. The Combinator S. *Information Computation*, 159(1–2):2–21, 2000.
- 16 Johannes Waldmann. Compressed loops (draft).
- 17 H. Zankl and A. Middeldorp. Nontermination of String Rewriting using SAT. 2007.
- 18 Harald Zankl, Christian Sternagel, Dieter Hofbauer, and Aart Middeldorp. Finding and certifying loops. In *Proc. Conf. on Theory and Practice of Computer Science (SOFSEM 2010)*, volume 5901 of *Lecture Notes in Computer Science*, pages 755–766. Springer, 2010.